

EARLY CREATION OF CROSS TOOLKITS FOR EMBEDDED SYSTEMS

Nikolay Pakulin npak@ispras.ru

Vladimir Rubanov vrub@ispras.ru

Institute for System Programming

Russian Academy of Sciences (ISPRAS)

Presentation Overview

- Introduction to ISPRAS
 - IRPRAS R&D directions in NESTER area
- Early Creation of Cross Toolkits ...
- Ideas for Joint R&D Projects

Description of the Institute

- Institute for System Programming Russian Academy of Sciences (ISPRAS) www.ispras.ru
- ISPRAS was founded in 1994.
- The Institute has more than 120 highly skilled researchers and software engineers,
 - 11 of them have Doctor degree,
 - over 40 of them have Ph.D. degree.
 - Many employees of the Institute are professors in classical and technical universities at the same time.
- Academician Victor Ivanninkov is a founder and Director of ISPRAS
- Rich history of international cooperation
 - Intel, IBM, Microsoft, Nortel, Telelogic ...
 - Gelato, FP6/FP7 ...

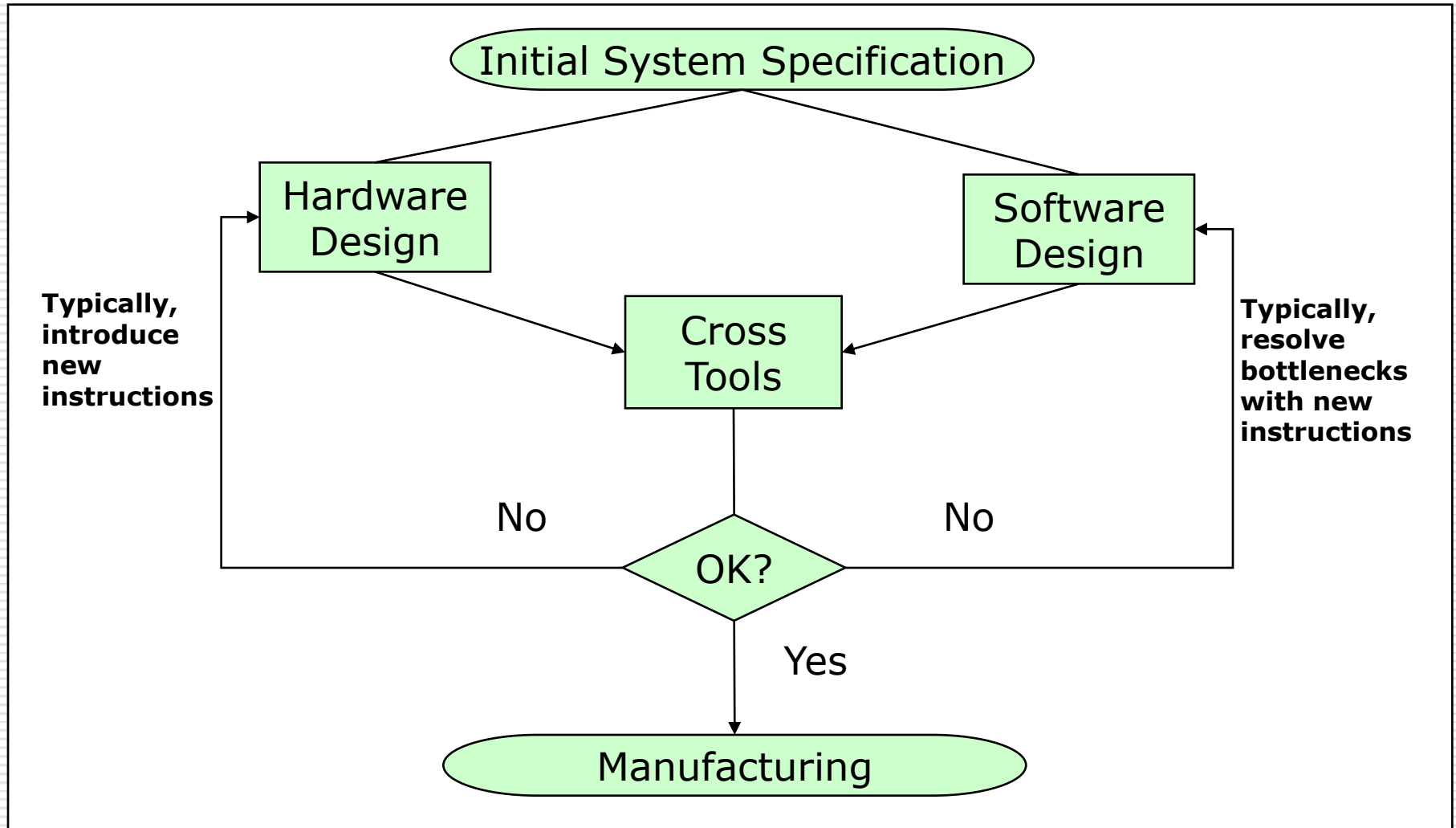
Relevant Research Topics

- Software engineering foundations; Verification and validation technologies; Model driven design and testing
 - Linux Verification Center (Linux Foundation largest contractor)
 - Verification and certification of mobile and embedded Linux distributions and applications
- CPU module and system testing
 - Automated construction and controlled execution of test assembly programs
 - Cross development tools for embedded systems; Operating systems
- Automation of cross-toolkit construction
 - MetaDSP framework with multiple industrial applications
- Power consumption optimization
 - GCC optimizer for ARM CPUs

Scope of the Work

- Embedded system: HW/SW system with a specific purpose and tough constraints on available resources (CPU, memory, power)
 - Control modules, smart objects, sensors
 - Digital signal processing (audio/video and imaging appliances)
- Cross toolkit: enables SW development on a workstation rather than the target HW
 - Building tools: C compiler, assembler, disassembler, linker, firmware generator.
 - Execution tools: debugger, simulator, profiler.

Using Cross Tools in Embedded System Design



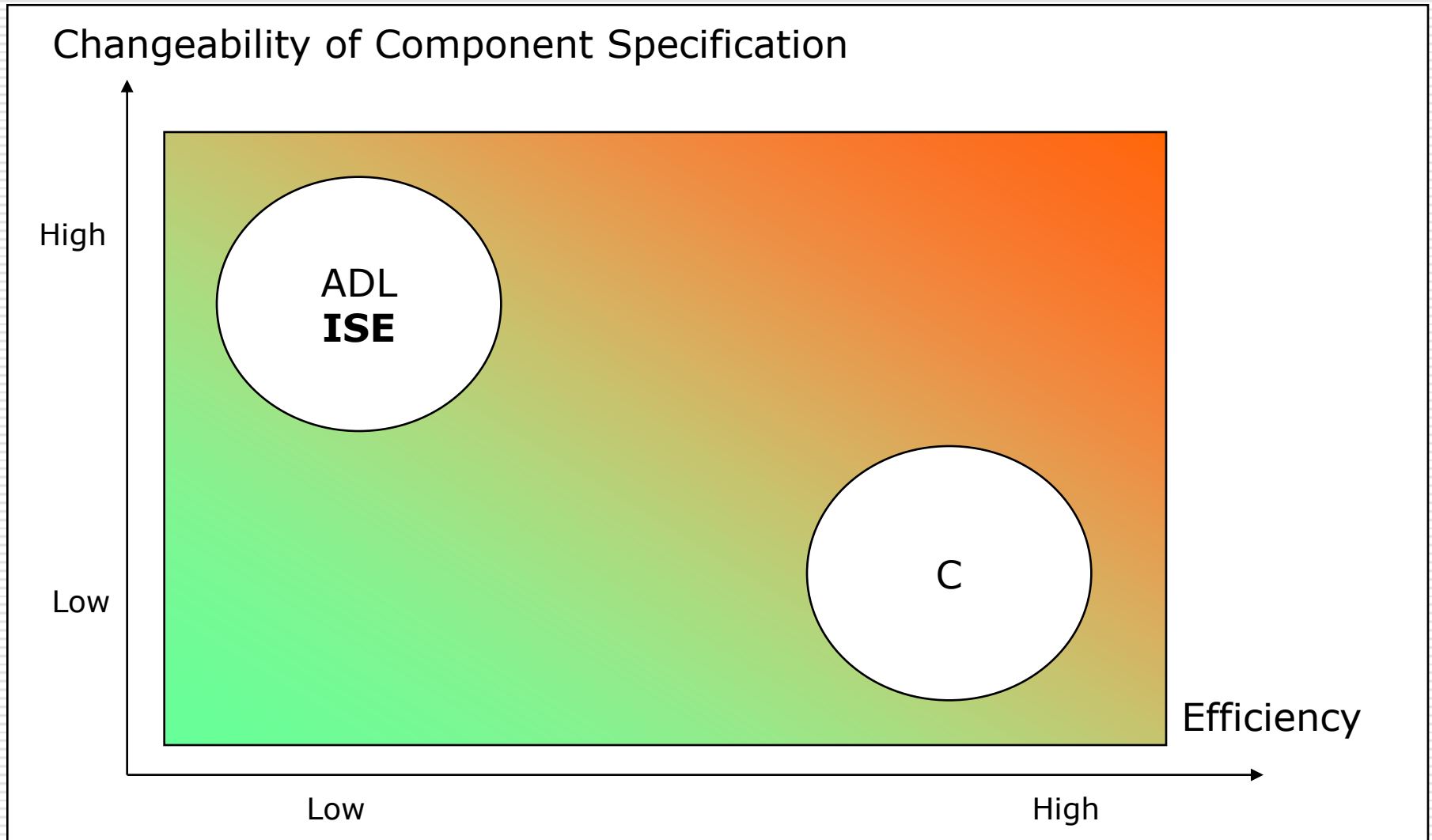
Requirements for Cross Tools Development

1. Complete cross toolkit:
 - Building tools: C compiler, assembler, disassembler, linker, firmware generator.
 - Execution tools: debugger, simulator, profiler.
2. Cycle-accurate simulation and profiling
3. High performance
 - e.g. 10 MCPS on 2 GHz PC
4. Availability at early development stages
5. Quick response to design changes
6. Method's concepts and notations should look familiar to industrial developers

“Pure Specification” Approaches

- HDL (VHDL, SystemC)
 - Extremely low simulator performance
 - Late stage in the development process
 - No explicit instruction set specification – only simulator available
- ADL (nML, EXPRESSION, ISDL, ...)
 - No cycle-accurate simulation
 - Low simulator performance
 - Unfamiliar for engineers
- C/C++
 - Low responsiveness to instruction set changes

Hybrid Specification: Changeability vs. Efficiency



Primary Components

Stable.
Language: C

- Pipeline control
 - Pipeline state
 - Stage transition
 - Instruction load
- Traps, interrupts

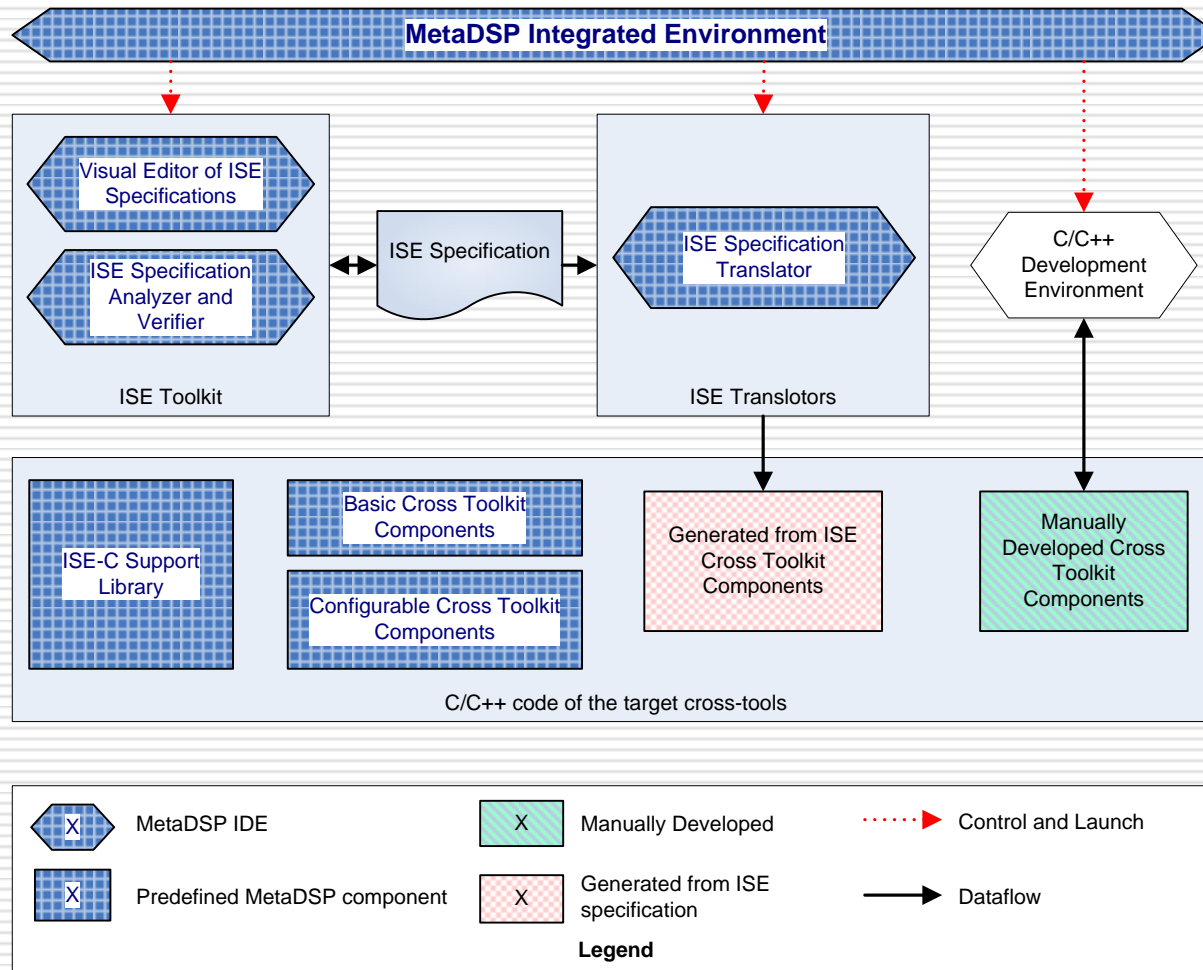
ISE-C

Instruction execution

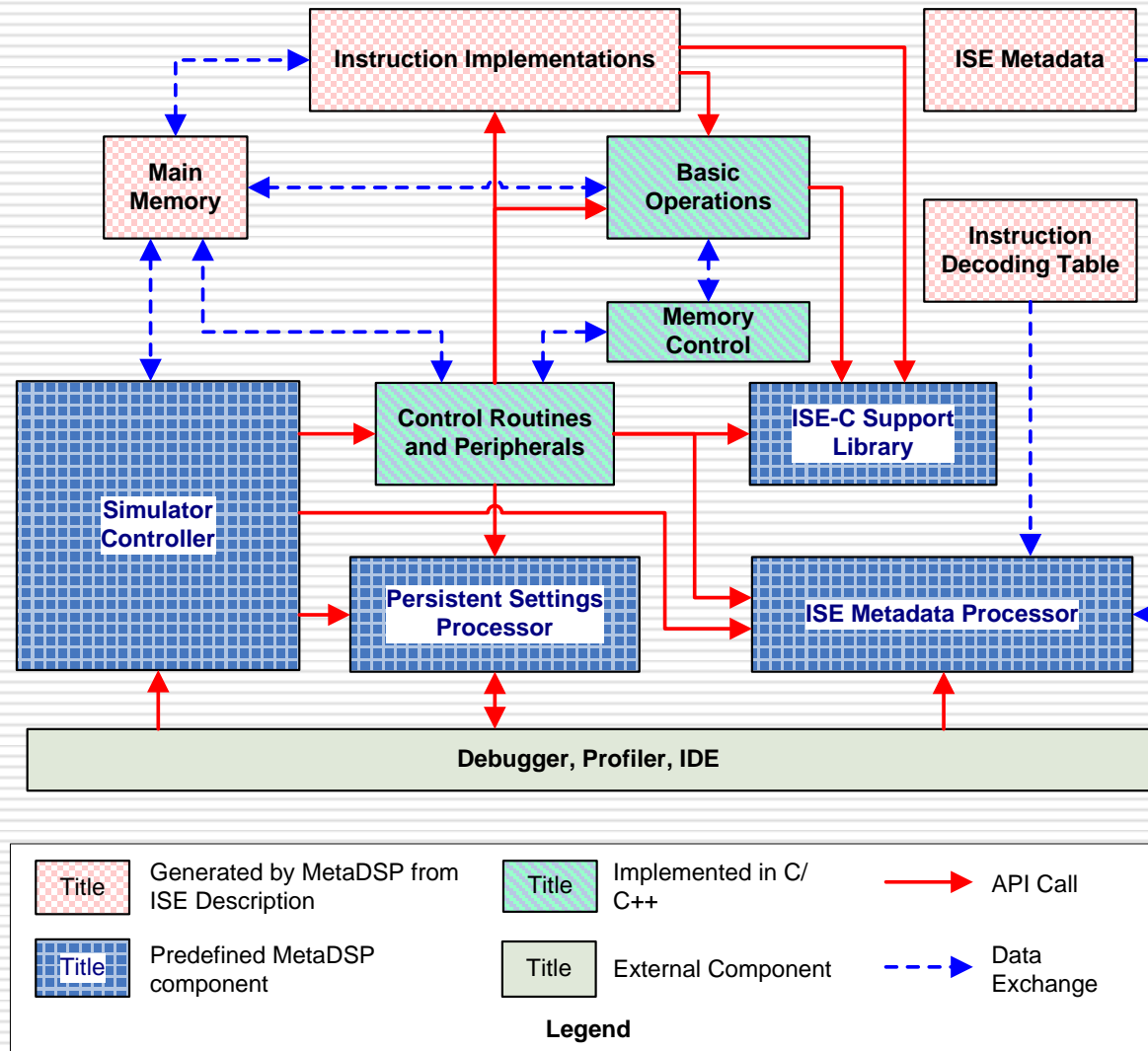
Volatile.
Language: ISE (Instruction Set Extension)

- Instruction set
 - Binary encoding
 - Dependencies and conflicts
- Memory and registers
- Resources

MetaDSP Framework



MetaDSP: Simulator Architecture



OSCAR IDE: Execution Environment

The screenshot displays the OSCAR IDE interface during a debugging session. The main window shows the source code for `encoder_decoder_vad1_full_rt - OSCAR Studio [break] - [c2_11pf.c]`. The code is annotated with red numbers 1 through 5, corresponding to the red callouts in the other panels.

1 Project Explorer: Shows the file structure of the project, including `Encoder` and `Common` folders.

2 Source Code: The current file is `c2_11pf.c`. The code includes a loop: `for (i1 = ipos[1]; i1 < ipos[2]; i1++)`. The current execution point is at `alp1 = L_mac(alp1, ipos[i1]);`.

3 Call Stack: Shows the current call stack with the top frame being `encoderTask()` at PC `0x9337`.

4 Registers: Shows the state of registers. `GR2` is highlighted with value `448F`. The format menu is open, showing `Hexadecimal` selected.

5 Disassembly: Shows the assembly code for the current instruction: `[65E0] 0614FD MOV GR4, DMO(FP-3)`.

Memory: Shows a memory dump for `DMO` with values like `0000 0000 1E13 C3DA 1E13`.

Watch: Shows a watch list with expressions and values, including `lag_h` with value `const int * 0x05F0`.

Tasks: Shows the task list with `encoderTask` in the `RUNNING` state.

Profiler: Shows the execution profile with `cod_amr` being the most time-consuming function, accounting for `64.0%` of the total execution time.

At the bottom, the status bar indicates: `User Cycles: 9 395 827 Cycles: 9 395 827 Samples: 0 EXE:0:65E0h Ln 228, Col`

Industrial Applications

- **50** revisions of cross tools for Freehand Platf. 2; 2001, Freehand DSP AB (Sweden).
- **45** revisions of cross tools for MicroDSP 1.0; 2002-2003, VIA Technologies (Taiwan).
- **27** revisions of cross tools for MicroDSP 1.1 and extensions (DFFT, ALBI bus) ; 2003-2004, VIA Technologies (Taiwan).
- Cross tools for stable ZAC CPU; 2004, VIA Cores (USA).
- **33** revisions of cross tools for VIA DSP 2 variations and extensions; 2005, VIA Technologies (Taiwan).

ISE-Driven Cross Tools Development

- Application Domain:

YES: modern 16/32 bit DSP, 32-bit ARM

NO: contemporary general-purpose CPU

- Efficiency compared to manual C/C++
Cross Tools Development

Specification size	Reduced by a factor of 12
Development Efforts	Reduced by 60%
Responsiveness to Instruction Set changes	Improved by a factor of 10

Cooperation Sought: Open Source

- Open-source framework for embedded systems design & development,
 - covering:
 - HW: from design to production specification
 - SW: from design to implementation
 - Unit, module & system V&V
 - Partners search:
 - Industrial partners
 - Software developers
 - Tool vendors
- Integration into existing commercial frameworks
 - Partners search:
 - Tool/framework vendor



ISE Language Design Goals

- ISE specification structure should be aligned to “Instruction set reference” style
- Efficient support for irregular instruction encoding
 - Diverse encoding formats for different instructions
- C-like notation for functional specification

ISE Language Features

- Global architecture specification
 - Number of pipeline stages
 - CPU resources (buses, shift modules, accumulators, etc.)
- Memory and registers specification
 - Register pool
 - Memory banks, memory regions
- Operand definition
 - Register operand
 - Memory operand
 - Indirect addressing
- Instruction specification
 - Binary encoding
 - Cycle-precise; multi-stage operations
- Inter-instruction conflicts specification

Memory Specification

.storage

```
GR WORD16 [0..7] // 8 16-bit registers
AR WORD16 [0..3] // 4 address registers
ACR NBIT<36> [0..1] // 2 36-bit accumulators
internal alu_temp WORD32 // 32-bit ALU accumulator
internal mul_res WORD32 // 32-bit internal MAC
PREG PC unsigned int // program counter
// program memory (256 Kwords)
PMEM PM WORD16 [0..2**18-1]
// 2 banks of data memory
DM {
    DM0 WORD16 [0..65535] // main memory(64 Kwords)
    TM0 WORD16 [USERDEF] // DFFT coefficients
}
```

Operand Types

.otypes

// General purpose register

GRs {GRn : SSS}

// Constant embedded in instruction code

C4 {const4 : CCCC}

// Memory bank identifier

DMx {DMa : X}

// Indirect addressing

DM_GRs_C4 {DM_GR_offset : M-SSS-CCCC}

Instruction Specification

.instructions

```
<MOV002> MOVE {DMa:M}({GRs}), {GRt}
```

```
00M0-0000-1SSS-1TTT
```

```
"MOVE DMa(GRs), GRt"
```

```
properties [ rgrn:GRs, rgrn:GRt ]
```

```
action {
```

```
    DMa[GRs] = GRt;
```

```
}
```

```
<MAC01> MAC {ACRa}, {GRs}, {GRt}
```

```
0111-00A0-1SSS-1TTT
```

```
"MAC ACRa, GRs, GRt"
```

```
properties [ racr:ACRa, wacr_2:ACRa,
```

```
                rgrn:GRs, rgrn:GRt ]
```

```
resources [ EXE_MAC(mac) ]
```

```
action {
```

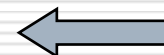
```
    mul_res = GRs * GRt;
```

```
}
```

```
action:EXE_MAC {
```

```
    ACRa = ACRa + mul_res;
```

```
}
```



Syntax Rule pattern



Encoding



Human readable mnemonics



Properties



Behavior

Conflicts specification

■ Errors

(I1:P:wacr_2 == I2:P:wacr)

:: error "ACR conflict on write"

■ Warnings

(I1:R:mac && I2:P:clr)

:: warning "Clear instruction after MAC instruction, result unspecified"

Toolkit Support

■ MetaDSP Toolkit:

- ISE specification analysis and verification tools
- ISE translator
- Reusable software components
 - Simulator core modules
 - Ready-to-use configurable modules for cross-tools construction

■ OSCAR IDE:

- Workbench for application developer
- C/C++/Assembler app development
- Debugging
- Simulation / profiling